EV052704359

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Request Persistence During Session Authentication

Inventor:
Baskaran Dharmarajan

ATTORNEY'S DOCKET NO. **MS1-1055US**

TECHNICAL FIELD

This invention relates to client-server sessions and, in particular, to maintaining a request during session authorization.

BACKGROUND

Many web sites implement security by way of encrypted authentication tokens placed on the client as browser cookies. Many of the sites have a session time, after which the authentication tokens become invalid and require that the user re-submit a password to restore the session. This can be very disruptive to the user. For example, a user composing an email message will not be aware that the authentication token has expired. When the user subsequently attempts to submit the email message, the user will be prompted with a "Re-enter your password" page. Not only will the disruption preclude the email message from being sent, but the user will also have to re-create the email message.

To decrease the potential for such a disruption, some web sites have implemented "rolling time windows." This means, if a user's session time is two hours, and there is no session activity for the two hours, then before the next request is processed, the user will be prompted to re-authenticate by providing a password again. However, this approach poses a security problem. It is possible for someone to steal a user's cookies containing the encrypted authentication tokens, and thus, the user's identity. The thief can then infinitely maintain the user's identity by simply executing a script to automatically refresh a page or perform some other session activity at regular intervals that are less than the session time-out period.

SUMMARY

Systems and methods for keeping a request persistent during session authentication are described. A session authentication system verifies that a client has been authenticated before processing a request submitted by the client. When the system receives a request from a client and cannot verify that the client has been authenticated, the system maintains the request and re-directs the client to obtain valid authentication. Upon verification that the client has been authenticated, the system processes the maintained request.

BRIEF DESCRIPTION OF THE DRAWINGS

The same numbers are used throughout the drawings to reference like features and components.

Figure 1 illustrates example communications between a client computer system, a login server, and an application server, to perform session re-authentication.

Figure 2 is an example block diagram of a session re-authentication system implemented in an application server.

Figure 3 is an example flow diagram of session authentication performed by a session re-authentication system.

Figure 4 is an example flow diagram of client redirection performed by an example session re-authentication system given an example process request.

DETAILED DESCRIPTION

The following describes systems and methods that persist, or otherwise maintain, a client request during session authentication. A re-authentication

system implemented on a server computer system provides increased security without disrupting user workflow in a client-server environment. When a request is submitted from the client to the server, the re-authentication system verifies that the session is secure. If the re-authentication system cannot verify that the session is secure, the system persists (*e.g.*, saves or maintains) the request and directs the client to re-authenticate the session. When the client session is re-established, the re-authentication system directs the server to process the saved request, instead of requiring that the request be re-submitted from the client.

For example, an application server may host an Internet-based email application. Users may log in with a username and password, and then send and receive email over the Internet. Without a session re-authentication system in place, a user may lose an unsent email message if the user's login expires while the user is composing the message, but before the user sends the message. On the other hand, if the application server that hosts the Internet-based email application is implemented with a session re-authentication system, the message is not lost. If the user's login has expired, the unsent email message is saved and the user is given an opportunity to login again. After a successful login, the saved email message is automatically sent, as previously requested by the user.

Figure 1 illustrates example communications between devices to perform session re-authentication in an authentication environment 100 which includes a client computer system 102, a login server 104, and an application server 106. Although Figure 1 depicts the login server 104 and the application server 106 as physically separate systems, it is recognized that functions performed by the two devices may be performed by the same server system.

Client computer system 102 submits login request 108 to login server 104. After verifying the client identity (*e.g.*, a submitted username and password), login server 104 provides client computer system 102 with authentication token 110, such as a cookie, to be stored on the client. After obtaining a valid authentication token, client computer system 102 submits process request 112 to application server 106. The process request may be, for example, a request to send an email message composed using an Internet-based email application maintained on the application server. Application server 106 verifies that the session between client computer system 102 and application server 106 has been authenticated, and then processes the process request. For example, application server 106 may verify that the session has been authenticated by examining the client's authentication token. In one implementation, a copy of the authentication token may be sent to the application server as part of the process request. In another implementation, the application server may access the authentication token stored on the client computer system to determine whether or not the authentication token is valid.

To enhance session security, authentication token 110 provided by login server 104 may expire after a period of time. For example, a session may be considered no longer secure if two hours have passed since the client last obtained an authentication token from the login server.

When application server 106 receives process request 112 from client computer system 102 and is unable to verify that the client-server session has been authenticated (*e.g.*, the client's authentication token is missing, has expired, or is otherwise not valid), application server 106 saves process request 112 as pending request 114 in a pending request store 116. Application server 106 then generates and sends an authentication redirect 118 to client computer system 102.

Authentication redirect 118 contains information that will direct client computer system 102 to request a valid authentication token and also contains a return address 120 that is associated with an address 122 of stored pending request 114.

In one implementation, return address 120 is a URL (uniform resource locator) and is generated based on address 122, which is associated with pending request 114. Authentication redirect 118 directs client computer system 102 to send a second login request 108 to login server 104. As described above, after verifying the client identity (*e.g.*, a submitted username and password), login server 104 provides client computer system 102 with a valid authentication token 110. After client computer system 102 obtains the valid authentication token 110, authentication redirect 118 directs client computer system 102 to access the stored pending request 114 through return address 120. Application server 106 then verifies the client's authentication token 110 and processes pending request 114.

Figure 2 shows an exemplary session re-authentication system 200 implemented at application server 106, which includes a processing unit 202 and a memory 204. The session re-authentication system 200 is implemented in memory 204 along with one or more application program(s) 206. Processing unit 202 processes requests from the client, either when a process request is received and the client authentication token is deemed valid, or after a process request is saved and a valid authentication token is subsequently verified.

Session re-authentication system 200 includes a client interface 208 that facilitates communication between session re-authentication system 200 and client computer system 102 using a communication protocol (*e.g.*, HTTP). Session re-authentication system 200 also includes an authentication token verifier 210, an authentication redirect generator 212, and the pending request store 116.

The authentication token verifier 210 verifies that that the client has a valid authentication token when the client submits a request. Authentication token 110 is provided to client computer system 102 by login server 104 (Figure 1). The verification performed by authentication token verifier 210 may include verifying that the username associated with authentication token 110 matches the username associated with client computer system 102. Additionally, authentication token verifier 210 may verify that authentication token 110 has not expired based on an expiration time assigned by login server 104 and associated with authentication token 110. Authentication token verifier 210 may also use other criteria to determine whether the authentication token 110 is valid. Examples of other criteria that may be used to determine whether the authentication token is valid include, whether the account is associated with previous violations of terms of a service agreement; whether the account is associated with a minor, and in such a case, whether the minor has sufficient permission from a parent; and whether the account has been terminated due to lack of use over a period of time, and therefore needs to be recreated.

The pending request store 116 stores process request 112 as pending request 114 when authentication token verifier 210 determines that the authentication token associated with the client is invalid. In one implementation, the pending request may be stored as a file. In an alternate implementation, the pending request may be stored as a database file. In another alternate implementation, the request may be encoded and persisted as a set of cookies if the total size does not exceed the protocol limits, thus saving back-end write-read-delete round trips for a 3 tiered internet system utilizing file or database based back-ends. A stored pending request 114 is associated with the address 122

that is used to generate return address 120 as part of a redirect operation described below.

The authentication redirect generator 212 creates authentication redirect 118 to instruct the client to obtain a valid authentication token and access the stored pending request. Authentication redirect 118 directs client computer system 102 to submit login request 108 (Figure 1) to login server 104. After the login server provides authentication token 110, the authentication redirect 118 uses return address 120 to direct the client to access the address 122 associated with stored pending request 114, which in turn causes application server 106 to process stored pending request 114.

Figure 3 illustrates a session authentication process. The session authentication process is illustrated as a set of operations shown as discrete blocks. The process may be implemented in any suitable hardware, software, firmware, or combination thereof. The order in which the operations are described is not to be construed as a limitation.

At block 302, a session re-authentication system receives a process request from a client. For example, re-authentication system 200 (implemented as part of application server 106) receives process request 112 from client computer system 102 (Figure 1). The process request is received through client interface 208 of re-authentication system 200 (Figure 2).

At block 304, the re-authentication system determines whether the client has a valid authentication token. For example, authentication token verifier 210 of session re-authentication system 200 determines whether an expiration time period associated with the client's authentication token has passed. Although this implementation is described with reference to a time-based expiration, it is

recognized that other criteria may be used to verify the validity of an authentication token, such as the version of a key used to encrypt the token. For example, if the login server currently encrypts authentication tokens using version three of an encryption key and the re-authentication system detects an authentication token encrypted with version two of the encryption key, the re-authentication system determines that the authentication token encrypted with version two of the encryption key is invalid.

If the re-authentication system determines that the client has a valid authentication token (*i.e.*, "yes" from block 304), the application server processes the received request at block 306. On the other hand, when the re-authentication system determines that the client does not have a valid authentication token (*i.e.*, "no" from block 304), the re-authentication system stores the received request at block 308. The request is stored, for example, in pending request store 116 described above with reference to Figure 2. As described above, an address is associated with the stored request.

At block 310, the re-authentication system redirects the client to the login server to obtain a valid authentication token. The redirection instruction is generated, for example, by authentication redirect generator 212 of session re-authentication system 200.

Based on the return address that is included as part of the generated redirect, the re-authentication system receives a request from the client to access the stored request at block 312. For example, client computer system 102 is directed by way of return address 120 to stored pending request 114 (Figure 1).

At block 314, the re-authentication system determines whether the client has a valid authentication token. If the re-authentication system determines that

the client does not have a valid authentication token, (i.e., "no" from block 314), the process continues as described above, at block 310. On the other hand, when the re-authentication system determines that the client does have a valid authentication token (i.e., "yes" from block 314), the application server processes the stored pending request, such as pending request 114.

Figure 4 illustrates a client redirection process performed by an example session re-authentication system given an example process request associated with an invalid authentication token. The process request of this example is a process request to send a composed email message. The request is directed to an email software application component on the application server. In this example implementation, the encrypted authentication token is sent to the application server as part of the request. When the session re-authentication system determines that the authentication token associated with the received request is invalid, the session re-authentication system begins the client re-direction process.

At block 402, the example session re-authentication system extracts information from the request that will be necessary to process the request at a later time. The session re-authentication system stores the extracted information in the pending request store. For example, given the example process request to send a composed email message, the name of the email application component to which the request was directed and the contents of the email message are stored in the pending request store. For example, the information extracted from the request is stored in a file on a storage device associated with the application server. The name of the file in which the request is stored may be randomly generated.

At block 404, a redirect generator component of the example session re-authentication system generates and encrypts a pending request location identifier. The pending request location identifier is associated with the file in which the information extracted from the process request is stored and contains the information necessary for the application server to locate the file at a later time.

At block 406, the redirect generator appends the encrypted pending request location identifier to a software component identifier. In this example, the software component identifier indicates the email software application component to which the original request was directed.

The application server is associated with an address, for example, a URL. At block 408, the redirect generator appends the software component identifier and encrypted pending request location identifier to the URL associated with the application server, generating a return URL that will direct the client back to the stored pending request.

As described with reference to block 310 of Figure 3, the session re-authentication server then sends a redirect instruction to the client, which includes the generated return URL as a parameter. The redirect instruction directs the client to the login server to obtain a valid authentication token. The login server authenticates the user and generates a valid authentication token.

At block 410, the session re-authentication system receives a request from the client to access the return URL. The request is in the form of the return URL with the valid authentication token appended to the end.

At block 412, the re-authentication system verifies that the received authentication token is valid.

At block 414, the session re-authentication system decrypts the pending request location identifier that is embedded in the return URL.

At block 416, the session re-authentication system directs the software component identified in the return URL (for example, the email software application component to which the original request was directed) to process the request stored at the location indicated by the decrypted pending request location identifier.

Conclusion

Although the systems and methods have been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.